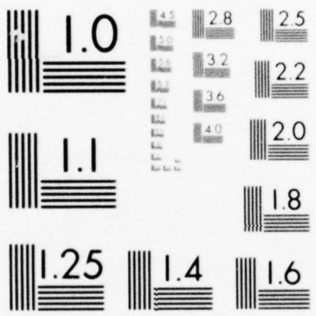
AD-A050 137 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 6/4
CELLULAR PYRAMIDS FOR IMAGE ANALYSIS.(U)
NOV 77 C R DYER AFOSR-77-3271
UNCLASSIFIED TR-596 AFOSR-TR-78-0126 NL

1 OF 1
AD
A050137

END
DATE
FILMED
3-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

AFOSR-TR- 78 - 0126

2

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
1807-1856-1920

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(2)

TR-596 November, 1977
AFOSR-77-3271

CELLULAR PYRAMIDS FOR IMAGE ANALYSIS, 2

Charles R. Dyer
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

In an earlier report, a new class of multilayer bounded cellular automata was defined and shown to improve the lower bound recognition time for many basic array recognition tasks. We continue our investigation of pyramid cellular acceptors by presenting new results on their capabilities as two-dimensional pattern-recognizing machines.

The support of the U. S. Air Force Office of Scientific Research, under Grant AFOSR-77-3271, is gratefully acknowledged, as is the help of Judy Myrick in preparing this report. The author also wishes to thank Professor A. Rosenfeld for helpful discussions and comments.

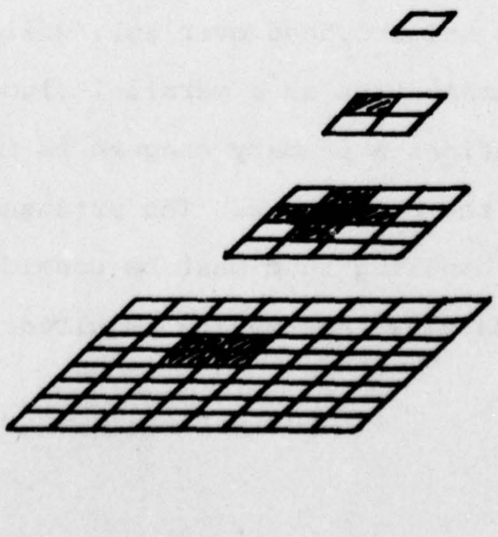
1. Introduction

A central concern of image analysis is to construct algorithms for extracting a description from a given picture. In general, this involves segmentation of the picture into parts, measurement of properties of the parts, and determination of relations among the parts. To this end, many basic picture analysis techniques have been developed. For example, thresholding, template matching, edge detection and connected component analysis are a few of these techniques.

The design and evaluation of efficient image analysis techniques and algorithms have not been adequately researched. In particular, the tradeoffs between computer architecture and algorithm complexity for a given problem are of considerable interest because of the high data rates necessary in many image application areas, while hardware costs continue to drop. Furthermore, many basic image analysis operations operate independently on each point of the image and its neighbors. Cellular parallelism is very efficient in computing such local operations. In fact, the most obvious technique computes each point's new value independently, leading to a very inefficient serial algorithm (due to neighborhood overlap), while being ideally suited for implementation as a parallel algorithm.

In parallel computations a primary concern is the distribution of results among the processors. The arrangement of the data and the processors handling them must be considered in light of complicated data flow that may be required. This problem is especially evident in cellular architectures, in which processors are uniformly interconnected to a fixed number of neighbor processors. Under conditions where communication between distant processors is necessary, unavoidable inefficiencies may arise.

With these factors in mind, we introduced in [1] a pyramid cellular machine, in which the processors, or cells, are configured in a logarithmically tapering pyramid. Each pixel in the input picture is associated with its own cell in the base of the machine. Each cell operates on its own local memory and has its own local control, which is globally synchronized by a discrete-step clock. Each cell can examine the memory contents of its nine neighbors -- its "father" cell in the layer above, its four nearest neighbors in its own layer, and its four "son" cells in a two-by-two block in the layer below. The model is defined by stacking cellular array automata, where each layer contains one quarter the number of processors in the layer below it.



In [1] this new model's computing power was compared with that of several others, and some basic pyramid recognition algorithms were described. It was shown that pyramid cellular acceptors can solve many nontrivial tasks in time proportional to the logarithm of the diameter of the input. This is in constrast to conventional bounded cellular acceptors which require at least time proportional to the diameter of the input for recognition. For the most part, however, the recognition results presented in [1] were confined to the one dimensional case, i.e., when the pyramid reduces to a triangle of processors and the input is a string of symbols.

This report extends the analysis of cellular pyramids by providing new results on recognizing two-dimensional picture languages. We first review the definition of a cellular pyramid in Section 2. Section 3 describes some basic pyramid recognition algorithms, including local property detection and counting, palindromes, rectangles and squares. In Section 4 we compare pyramid acceptors with finite-state acceptors and bounded cellular acceptors.

The cellular pyramids defined in this report are, in general, nondeterministic; but the algorithms given in this report are all deterministic. Some results on nondeterministic cellular pyramids will be presented in a subsequent report.

2. Definitions

In this section we review the definitions of a bounded cellular array acceptor, pyramid cellular acceptor, and bottom-up pyramid cellular acceptor. We also introduce, for comparative purposes in Section 4, the up-down pyramid cellular acceptor.

A bounded cellular array acceptor (CA) is a finite, rectangular array of identical finite state machines (FSM's), or cells. Each of these cells is a quadruple $M=(Q_N,Q_T,\delta,A)$, where Q_N is a nonempty, finite set of states, $Q_T \subseteq Q_N$ is a nonempty, finite set of input states, and $A \subseteq Q_N$ is a nonempty, finite set of accept, or final, states. $\delta:Q_N^5 \rightarrow 2^{Q_N}$ is the state transition function, mapping the current states of M and its four nearest neighbors into a set of possible next states for M. If M is deterministic, the range of δ is a single state from Q_N. Associated with the CA is a special boundary state $\# \in Q_N$ which can never be created or destroyed by δ.

A configuration of a CA is an assignment of states from Q_N to each cell in the CA. A step of computation consists of a single application of the state transition function δ simultaneously at each cell. An input configuration is a configuration before the first step. If a cell is on the border of the array its initial state must be the boundary state #, otherwise its state may be chosen from Q_T. Cells initially in the boundary state are called boundary cells;

all others are called retina cells. An input configuration is accepted by a CA if at some step the upper-left corner retina cell enters an accept state. The set of input configurations accepted by a given CA defines its language.

A <u>pyramid cellular acceptor</u> (PA) is a pyramidal stack of CA's, where if the bottom array has retina size 2^r by 2^r, then the next lowest array has size 2^{r-1} by 2^{r-1}, and so on, until the (r+1)st layer consists of a 1 by 1 CA. This apex cell in a PA is called the root. Each cell is an identical FSM $M=(Q_N,Q_T,\delta,A)$, where Q_N, Q_T, and A are defined as above. The transition function $\delta:Q_N^{10} \to 2^{Q_N}$ maps 10-tuples of states into sets of states. That is, each cell has nine neighbors -- four son cells in a two-by-two block in the layer below, four nearest neighbors in its own layer, and one father cell in the layer above. More precisely, let M be the (i,j,k)th cell in a PA, where M is in the ith row and jth column of retina cells in the kth layer from the bottom. Then M's father is cell $(\lceil i/2\rceil,\lceil j/2\rceil,k+1)$, where $\lceil x\rceil$ denotes the smallest integer greater than or equal to x. M's brother cells are (i-1,j,k), (i,j+1,k), (i+1,j,k) and (i,j-1,k), and its son cells are (2i,2j,k-1), (2i,2j-1,k-1), (2i-1,2j,k-1) and (2i-1,2j-1,k-1). The figure in Section 1 illustrates this neighborhood.

A configuration of a PA is an assignment of states from Q_N to each cell in the PA. An input configuration for a pattern of size 2^n by 2^n is an n+3 high stack of CA's. The bottommost layer is a size 2^{n+1} by 2^{n+1} CA, where every cell's initial state is the boundary state #. The next layer up is called layer 0 and is a size 2^n+1 by 2^n+1 CA, whose border cells have initial state #, while the other cells define the base array where the input pattern is "stored" -- that is, their initial states are chosen from Q_T and define the input image to be recognized. Layer 1 is a size $2^{n-1}+1$ by $2^{n-1}+1$ CA, with retina cells initialized to the quiescent state $b \in Q_T$. Layers 2 through n are similar to layer 1 except each layer's retina has one-quarter the number of cells in the layer below it. Above layer n is a single cell with initial state # which is the father cell of the apex cell in layer n.

The height of a PA is the length of the shortest path from the root to a cell in the base array. Thus a PA with base size 2^n by 2^n has height n and $4^n+4^{n-1}+...+1 = (4^{n+1}-1)/3$ retina cells. An input image is a 2^n by 2^n array of states from Q_T for some $n \geq 0$, which defines the initial states of the retina cells in the base array (layer 0). A step of computation consists of a single state transformation performed simultaneously at each cell. An input configuration is accepted by a PA if the root enters an accept state after a finite number of steps. The language accepted by a PA is the set of all input images accepted by it. (As with CA's, the language is defined for all legal input sizes and is <u>not</u> associated with a particular instance of a PA which has a specific height and recognizes only images of a specific size.)

A bottom-up pyramid cellular acceptor (BPA) is a PA whose state transition function is modified to be $\delta : Q_N^5 \rightarrow 2^{Q_N}$. In this simplification of PA's, the next state of a cell depends only on the current states of that cell and its four sons. Hence information can only move up the pyramid. All other aspects of the PA hold for a BPA.

Finally, we define a second simplification of PA's which gives another alternative neighborhood definition for restricting information transmission through a cellular pyramid. An up-down pyramid cellular acceptor (UDPA) is a BPA whose state transition function is defined as $\delta : Q_N^6 \rightarrow 2^{Q_N}$. Here the next state of a cell also depends on the state of its father, so that state information can move up and down, but not sidewise, through the pyramid.

The purpose of investigating these simplifications of the original model of computation is not because of any hardware considerations for simplifying the interconnection links. Rather, it is intended as an aid in studying the tradeoffs between neighborhood size, time and space (state set cardinality) bounds. By comparing variations of the model we gain insight into how computing power is affected by incrementally adding more channels for information flow.

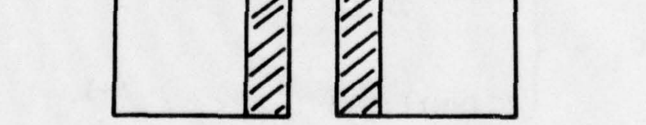
3. BPA Algorithms

This section considers several basic tasks for BPA's, including detecting and counting arbitrary local patterns, and recognizing palindromes, rectangles and squares.

We assume from now on that the size of the pyramid base is 2^n by 2^n, so that the height is n. The layers will be numbered 0,1,...,n starting from the base. To simplify the exposition, BPA's will not be defined in terms of states and transition functions; rather, we will specify algorithms in terms of transmitting and receiving information between son and father cells.

3.1 Local Property Detection

In this section we consider the problem of detecting the presence or absence of a d by d pattern* in the input of a BPA. The difficulty of this problem is that such a pattern can be badly positioned with respect to the pyramid so that only cells at an unbounded height above the base can see all of it. Furthermore, since a d by d block of the base must be matched against the given pattern at the lowest common ancestor of its d^2 base cells (because this cell may be the apex cell of the BPA), the number of matchings to be performed by a cell increases exponentially with its height. That is, a cell C in layer k must check all those d by d blocks containing base cells which extend across the "cracks" between C's sons' bases. The figure below illustrates the base cells of C which must be considered in the matching process at this cell. It



*Since any arbitrary finite pattern can be padded with "don't care" symbols to obtain an equivalent square pattern, there is no loss in generality in making this restriction.

is easily verified that there are $2(d-1)2^k - 3(d-1)^2$ distinct d by d blocks to be checked by C.

A one-dimensional solution to this problem was given in [1], but that solution does not generalize to two dimensions. In this section we present a two-dimensional solution.

Let C_i denote the first (leftmost) d-1 symbols in cell C's ith row, and let C_i' denote the last (rightmost) d-1 symbols in C's ith row. First, construct the BPA so that cells in layer k count modulo 2^k and output alternating length-2^{k-1} sequences of 0's and 1's [1].

Suppose cell C at height $\delta = \lceil \log d \rceil$ repeatedly receives a copy of the entire base beneath it and

a) decides whether or not the pattern is entirely contained with it.

and b) modulo 2^δ starting at time δ outputs C_i and C_i' for $1 \le i \le 2^\delta$.

Then a cell C in layer $\delta+1$ can use its modulo $2^{\delta+1}$ counters and the outputs of its four sons to compute its own C_i and C_i' values (modulo $2^{\delta+1}$), since

$$C_i = \begin{cases} [C(NW)]_i & , 1 \le i \le 2^\delta \\ [C(SW)]_{i-2^\delta} & , 2^\delta < i \le 2^{\delta+1} \end{cases}$$

and

$$C'_i = \begin{cases} [C(NE)]'_i & , 1 \le i \le 2^\delta \\ [C(SE)]'_{i-2^\delta} & , 2^\delta < i \le 2^{\delta+1} \end{cases}$$

where C(NW) is C's northwest son, etc.

Simultaneously, C can decide whether or not the pattern appears anywhere within its base as follows. Since we need only check d-by-d blocks which are not entirely contained in one of C's son's bases, we need only reconstruct the $2^{\delta+1}$ by 2(d-1) vertical band centered on the middle column of C's base in order match across the vertical crack. In fact, only a d by 2(d-1) running window of the band needs to be stored by cell C at any time. Using its modulo $2^{\delta+1}$ counter and its four sons' outputs, C can scan its vertical band since the ith row of the band is computable at time $\delta + i$ (modulo $2^{\delta+1}$) by

$$[C(NW)]'_i \| [C(NE)]_i \quad \text{if } 1 \le i \le 2^\delta$$

or $$[C(SW)]'_{i-2^\delta} \| [C(SE)]_{i-2^\delta} \quad \text{if } 2^\delta < i \le 2^{\delta+1}$$

where $\|$ denotes concatenation. After time d+δ C can begin checking whether or not the entire pattern is present anywhere in its d by 2(d-1) window.

In the same way C can simultaneously scan the $2(d-1)$ by $2^{\delta+1}$ horizontal band centered on the middle row of C's base in order to match across the horizontal crack. (This means that the cells at height δ will also have to output the top and bottom $d-1$ symbols in the ith column, $1 \leq i \leq 2^{\delta}$, modulo 2^{δ}.) If at any time the pattern is found in either band, a success signal is immediately sent up the BPA to the apex cell.

Similarly, each cell in layer $k > \delta+1$ behaves like the cells in layer $\delta + 1$, sequentially scanning their length 2^k vertical and horizontal bands starting at time k. Hence, if the pattern is detectable in layer k then the apex cell will know this at a time no later than $k + 2^k + (n-k) = 2^k+n$. In the worst case the pattern is detectable only by the apex cell, and time $2^n t_a(d) + nt_b(d)$ is required, where $t_a(d)$ is the time needed to update two d by $2(d-1)$ windows and search for the d by d pattern within them, and $t_b(d)$ is the time needed to copy four length d vectors.

If we assume that the d by d pattern occurs at least once and is equally likely to appear anywhere in the 2^n by 2^n base and $d << n$, the O (diameter) worst case bound given above reduces to O(log diamater) time in the average case. That is, there are $4^n/4^k$ cells in layer k, and each checks $2(d-1)2^k - 3(d-1)^2$ d by d blocks for the presence of the pattern. Thus in layer k $(k > \log d)$ a total of $2^{2n}[(2(d-1)2^k - 3(d-1)^2)/2^{2k}]$ d by d blocks are inspected.

It is easily verified that after processing the $(\delta+2)$nd layer over half of the 2^{2n} blocks have already been checked. Since the apex cell will know whether or not the pattern is detected in any of these layers by time $4d\ t_a(d) + n\ t_b(d)$, the O(log diameter) average time bound results.

3.2 Local Property Counting

Counting the number of occurrences of some particular symbol z in a BPA's base is an immediate generalization of the one-dimensional counting algorithm [1], which required O(log diameter) time. The problem of counting arbitrary local patterns is more difficult because cells at all levels must detect and count instances of the pattern and then add their counts in with all other cells' counts.

First, we show that a BTA can count the number of occurrences of an arbitrary local property in 2 log diameter time steps. That is, the apex cell will output, at time steps n through 2n, the n+1 bits in the binary representation of this number, least significant bit first.

Suppose that the desired pattern has length $d=2^\delta$. Each cell in layer δ receives at time δ a copy of the entire portion B of the base below it and outputs, at the end of step δ,

(1) a 1 if B matches the pattern, and a 0 otherwise;

(2) the initial d-1 length segment of B, call this string L;

and (3) the terminal d-1 length segment of B, call this string R.

Now consider a cell C in the (k+1)st layer. By the induction hypothesis, it receives from its sons at the ends of time steps k,...,2k the number of instances of the given pattern and the values of L and R in their base segments. Let L_1,R_1

be the values for C's left son, and L_2, R_2 the values for C's right son. Then $L_C = L_1$ and $R_C = R_2$ can be output by C at the end of time step k+1. In addition, $R_1 \| L_2$ is the length 2(d-1) segment centered on the middle of C's base. At most d-1 occurrences of the pattern can extend across C's mid-base cell and all the information for checking these d-1 positions is contained in the string $R_1 \| L_2$. Hence at time k+1 C can compute and store the number of times, S, that the pattern is found crossing C's midpoint and then add the least significant bit of S to the sum of the least significant bits of its sons' counts, which has just been computed. At step k+2 C computes its next least significant bit by adding the next bits from its sons and the next bit from S. This process is repeated at steps $k+3, \ldots, k+\delta$, since S is a δ-bit value. For the remaining steps $k+\delta+1, \ldots, 2k+2$ C just sums the next least significant bits from its sons. Clearly C's outputs at steps $k+1, \ldots, 2k+2 = 2(k+1)$ are just the bits of the sum of its sons' counts and its own count, i.e., the number of times the given pattern occurs in C's base segment. The total time required by the algorithm is $2n\, t_a + nt_b(d)$, where t_a is the time needed to perform bit addition and $t_b(d)$ is the time required to count the number of matches of a length d pattern in a length 2(d-1) string.

This algorithm does not generalize to two dimensions because, as we noted in Section 3.1, the number of match tests to be performed at a cell is now unbounded. Hence a cell cannot store this number in order to bit-serially add this count in with the counts of its sons.

By allowing sidewise transmission in the base, counting arbitrary local patterns reduces to the problem of counting occurrences of a particular symbol, since the base array can detect and mark all occurrences in 2(d-1) steps. Similarly, other algorithms which require information from cells only a bounded distance apart can profit from sidewise transmission in the base, which requires only a bounded amount of time. For example, we can compute the Euler number of a binary image in O(log diameter) time on such an extended BPA because it is a property that is computable from measurements taken on 2 by 2 windows [2]. On the other hand, counting the number of connected components in a binary image using a parallel shrinking process [3,4] in the base array to mark components, and then using the upper layers of the pyramid to count these marks, provides no advantage, since the shrinking alone requires up to O(diameter) time.

3.3 Palindromes

In two dimensions the palindromes may be defined in alternative ways depending on the axis of symmetry chosen. Vertically symmetric palindromes are defined as those arrays which are symmetrical about their central column. That is, if the input symbols of a 2^n by 2^n array A are indexed in row-major order and $A(1,1)=A(1,2^n)$, $A(1,2)=A(1,2^n-1)$,..., $A(1,2^{n-1}-1)=A(1,2^{n-1})$,...,$A(2^n,2^{n-1}-1)=A(2^n,2^{n-1})$ then A is a palindrome.

Before presenting the algorithm we first show how a cell in layer k of a BPA can be made to count modulo ab^{ck+d} for arbitrary natural numbers a,b,c,d. Since $ab^{ck+d}=a(b^d)(b^c)^k=ef^k$ we need only show how a cell in layer k can have a modulo ef^k counter for constants e and f. We have shown previously [1] how such a counter is built when e=1 and f=2. In the general case, base cells are defined to output 1's every e time steps, and non-base cells output 1's every fth time that their sons' counters output 1's. Thus layer 0 (base) cells are modulo $ef^0=e$ counters. If cells in layer k are counting modulo ef^k, then readily layer k+1 cells' f-step delay makes these cells $f(ef^k)=ef^{k+1}$ counters.

Recognizing palindromes with a BPA, as in the one-dimensional case, is based on performing a fixed sequential scan of its base. This is implemented by having cells in the kth layer count modulo 4^k as described above. The counters can be easily modified so as to output 0 for the first 4^{k-1}

steps, 1 for the next 4^{k-1} steps, 2 for the next, and 3 for the last 4^{k-1} steps. If a father's counter is 0 it copies its NW son's value, if it is 1 it copies its NE son's value, if it is a 2 it copies its SW son's value, and if it is a 3 it copies its SE son's value. Thus a cell in layer 1 copies its sons' values at successive steps in the following order: $\begin{matrix}0 & 1\\ 2 & 3\end{matrix}$. A cell in layer 2 copies the values of its NW son for four steps, then its NE son for four steps, etc., so that it repeatedly copies the values of its base cells in the following order:

$$\begin{matrix}0 & 1 & 4 & 5\\ 2 & 3 & 6 & 7\\ 8 & 9 & 12 & 13\\ 10 & 11 & 14 & 15\end{matrix}$$

By induction, it follows that a cell in layer k copies the values of its base cells at times $k, k+1, \ldots, k+4^k-1$; and this process repeats modulo 4^k.

We can obtain a mirror image of the above scan by simply changing the order of scanning one's sons to: $\begin{matrix}1 & 0\\ 3 & 2\end{matrix}$. It is readily seen that running both scans simultaneously means the apex cell copies all pairs of symbols in base cells which are vertically symmetric about the central column of the base. If all point by point comparisons match, then the BPA can accept its input. The total time required is $n+4^n-1$. This is nearly optimal, since it can be shown that the lower bound for recognition of palindromes on a BPA is $4^n/2$ by remarks similar to those in [1]. Alternative definitions of two-dimensional palindromes, for example by using horizontal or central symmetry, can be handled by essentially the same method. A related question which is still open is whether the apex cell of a BPA can copy the symbols in its base in raster order.

3.4 Rectangles and Squares

Let the language L_{RECT} consist of the class of input images which contain a single solid rectangle of 1's (with sides parallel to the sides of the base) on a background of 0's. Blum and Hewitt have shown [5] how an FSA on a two-dimensional tape can recognize L_{RECT} by checking each boundary point's local slope. Minsky and Papert [2] define a diameter-limited perceptron to recognize L_{RECT} by counting the local property "corner." That is, input image $B \in L_{\mathrm{RECT}}$ iff.

$$\text{(the number of occurrences of pattern } \begin{smallmatrix}1 & 0\\ 0 & \end{smallmatrix}\text{)} + \text{(the number of occurrences of pattern } \begin{smallmatrix}0 & 1\\ 1 & \end{smallmatrix}\text{)} \leq 4$$

where the patterns also include all 90° rotations of each.

A BPA is not well suited either for simulating the moves of an FSA (see Section 4) or for counting local properties (Section 3.2). [If we allowed the BPA to have sidewise transmission in its base, then the Minsky and Papert method could be implemented on a BPA and L_{RECT} would require O(log diameter) time to be recognized.] We now show how a BPA can recognize L_{RECT} by a third method which compares consecutive row cross sections of the base.

Each row of an input image contained in L_{RECT} must be of the form $0^r1^s0^t$. We shall now prove that each cell C in layer k can output at time steps k to 2k the k+1 bits in the binary representations of r,s, and t for the top row and u,v, and w for the bottom row in C's subbase. Associated with the first (least significant) bit of each number will be a flag indicating whether or not the value is zero.

Clearly the cells in layer 1 can compute this information. Now consider a cell C in the (k+1)st layer. By the induction hypothesis, it receives from its sons at the ends of time steps k,...,2k the values of r,s,...,w in their base segments. Let the subscripts 1,...,4 denote the northwest, northeast, southwest, and southeast sons of C, respectively. Then C's base has the following form

$_0r_1\ \ _1s_1\ \ _0t_1$	$_0r_2\ \ _1s_2\ \ _0t_2$	row 1
$_0u_1\ \ _1v_1\ \ _0w_1$	$_0u_2\ \ _1v_2\ \ _0w_2$	row 2^k
$_0r_3\ \ _1s_3\ \ _0t_3$	$_0r_4\ \ _1s_4\ \ _0t_4$	row 2^k+1
$_0u_3\ \ _1v_3\ \ _0w_3$	$_0u_4\ \ _1v_4\ \ _0w_4$	row 2^{k+1}

so that C computes its own r, s and t values as follows:

1) if $s_1=0$ and $s_2=0$ then $r=r_1+r_2$, $s=0$, $t=0$
2) if $s_1\neq 0$ and $s_2=0$ then $r=r_1$, $s=s_1$, $t=t_1+r_2$
3) if $s_1=0$ and $s_2\neq 0$ then $r=r_1+r_2$, $s=s_2$, $t=t_2$
4) if $s_1\neq 0$ and $s_2\neq 0$ and $t_1=0$ and $r_2=0$ then $r=r_1$, $s=s_1+s_2$, $t=t_2$

If none of these conditions holds, a failure signal is immediately propagated up to the root. The addition is performed as in the BTA counting algorithm (Section 4.2 of

[1]) where C functions as a bit serial adder. Similarly, C computes u, v, and w from $u_3, v_3, w_3, u_4, v_4, w_4$.

Simultaneously, C checks whether the two middle rows in its base, as specified by

$$0^{u_1} 1^{v_1} 0^{w_1} 0^{u_2} 1^{v_2} 0^{w_2}$$
$$0^{r_3} 1^{s_3} 0^{t_3} 0^{r_4} 1^{s_4} 0^{t_4} ,$$

match, are of the form $0^x 1^y 0^z$ and are consistent with its top and bottom rows. Say these two middle rows are of the form $0^{x_1} 1^{y_1} 0^{z_1}$ and $0^{x_2} 1^{y_2} 0^{z_2}$, respectively. Then, if $y_1 \neq 0$ and $y_2 \neq 0$, we must be considering two rows intersecting a potential rectangle. Consequently, only if $x_1 = x_2$, $y_1 = y_2$, and $z_1 = z_2$ can both rows be part of a legal rectangle. If $y_1 = y_2 = 0$, both rows are scanning only background; if only one of y_1 and y_2 is nonzero, we are seeing either the top or bottom of the rectangle. In these cases there is no matching to be performed, only a consistency check that prevents multiple rectangles from being accepted. Details will not be given.

By transitivity and the induction hypothesis we now know that all nonzero rows in C's base segment are connected and have the same description, since every consecutive pair of rows within C's subpyramid has been matched. If a cell ever detects a completed rectangle, a success signal is propagated to the apex cell which, if it receives exactly one such signal, accepts its input. The total time required by the algorithm is $2nt_a$ where t_a is time needed to perform one step of the serial addition process and the "sewing" together of quadrants by matching boundary rows.

We now indicate how a BPA can detect the fact that the rectangle of 1's in its base is a square. Define L_{SQ} to be a set of inputs containing a single solid square of 1's on a background of 0's. L_{SQ} can be recognized by an FSA which first checks that the input contains a single rectangle and then scans from this rectangle's upper-left corner at 45° toward its opposite corner. If the opposite corner is reached, the rectangle is a square, otherwise it is not [5]. L_{SQ} cannot be recognized by a diameter-limited perceptron which measures local properties. However, there does exist an order-limited perceptron for recognizing squares which works by stratification -- a process of sequentially enumerating and testing all possible starting positions and side lengths for a square in the image [2]. Again, neither of these methods is adaptable for use on a BPA.

A BPA can recognize L_{SQ} in O(log diameter) time by a modification of its own rectangle algorithm. Each cell C computes cross sections of its base segment's boundary rows and columns. In addition, if C's base is nonzero C computes a row and column cross section of the object contained in its base. This is done as follows: if C's top and bottom rows have just been computed to be all zeros and C's middle two rows are nonzero, then C's base bounds the height of the object. If the object is a rectangle, C outputs one of its middle rows, which is a row cross section of the rectangle. All cells above C just copy this cross section while computing their own boundary cross sections. Similarly, cells save a column cross section if a rectangle's width is bounded by their base segment.

When a cell detects a completed rectangle it simultaneously compares the 1's counts in the horizontal and vertical cross sections through the rectangle. If they are equal, a success signal is transmitted to the apex cell, otherwise a reject signal is sent. The apex cell accepts its input image if it receives exactly one accept signal and no reject signals at time step 2n. The total time required by this algorithm is $2n\,t_a$, where t_a is the time necessary to update six cross sections and "mend" together adjacent quadrants by matching their common boundaries.

Counting the number of rectangles in an input image by a BPA involves the same problem encountered in local property counting (Section 3.2). That is, there may be an unbounded number of rectangles which are detectable only by the apex cell.

The general technique of comparing consecutive row cross sections used in this subsection is adaptable for recognizing other fixed-orientation polygons whose boundary slopes are locally testable. For example, straight line segments with slopes which are multiples of 45° (with respect to the bottom edge of the image) are digitally realized as

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(18) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS

```
                x            x          x
 x x x        x              x           x
            x                x            x

  0°         45°            90°        135°
```

Therefore, a BPA can recognize 45° right triangles with one side parallel to a side of the base, or diamonds (squares rotated 45°), by comparing triples of consecutive rows. (Computing the description of the top and bottom two rows of each 2^k by 2^k block is sufficient for this test.) Similarly, straight lines at other fixed orientations relative to the base are characterized by periodic digital segments of bounded length. A BPA can be constructed to detect these segments' slopes by examining a bounded number of cross sections above and below each input row.

It is an open question whether a BPA can accept the set of binary images whose 1's form a convex subset. However, several necessary conditions for an image to be convex can be tested by a BPA. A binary image is row convex if it is connected and each row contains at most one connected component. This is a property that can be accepted by a BPA since it only requires verifying that there is at most one run of 1's on each row, and that the runs of 1's on any two adjacent rows overlap. Similarly, a BPA can recognize column or diagonal convexity.

4. The Power of PA's

In this section we investigate the power of PA's, BPA's and UDPA's by comparing their language recognition capabilities with those of CA's and FSA's. We show that PA's, UDPA's and CA's are all equivalent. We leave open the question of whether or not BPA's are stronger than FSA's on a two-dimensional tape.

4.1 PA's and CA's

In [1] we showed how a one-dimensional CA can simulate a one-dimensional PA, each simulation step requiring O(diameter) *time. This result immediately generalizes to two* dimensions. Clearly PA's can simulate CA's since only cells in the base array have boundary cells as sons. Thus all other cells can remain in the quiescent state while the base array copies the transitions of the CA using only state information from brother cells. If the CA's upper-left corner cell ever enters an accept state, the father of the upper-left corner cell in the PA's base array can detect this and begin propagating an accept signal to the root. Therefore, PA's are *equivalent to CA's.*

4.2 BPA's and FSA's

We have previously shown that BTA's are more powerful than FSA's, and can simulate them in O(log diameter)time. That result was aided by the fact that two-way nondeterministic FSA's are no more powerful than one-way deterministic ones. In two dimensions, however, it is know that placing restrictions on the allowable directions of motion does alter the power of FSA's. (Henceforth, FSA will mean 2-D FSA.)

An FSA on a two-dimensional input tape is a 5-tuple $M=(Q_N,Q_T,\delta,A,q_0)$ where, as in the one-dimensional definition, Q_N is a nonempty, finite set of states, $Q_T \subseteq Q_N$ is the set of input states (tape symbols), $A \subseteq Q_N$ is the set of accept states, and $q_0 \subseteq Q$ is the start state. The state transition function permits four directions of movement -- up, down, left and right. That is, $\delta:Q_N \times Q_T \rightarrow Q_N \times \{U,D,L,R\}$ in the deterministic case; $\delta:Q_N \times Q_T \rightarrow 2^{(Q_N \times \{U,D,L,R\})}$ in the nondeterministic case.

To begin with we will consider a very restricted type of FSA, namely one that can only do a fixed, "one-way" scan of its tape. In particular, we show how a BPA can simulate an FSA M that can only do a raster (row-major) scan of its input. Tapes which are not square and whose sizes are not powers of 2 can be padded with a special tape symbol ∅ at the right and bottom and the transition function altered so that M moves over ∅'s without changing states. We assume that M is deterministic, since clearly one-way nondeterministic FSA's are no more powerful, by the same arguments used to show equivalence in the one-dimensional case [6].

Let the state set of M be $S=\{s_1,\ldots,s_m\}$. Each cell in layer k can be made to count modulo 2^k [1] and this counter can be readily modified to act as follows: for the first 2^{k-1} steps it outputs 0, for the next 2^{k-1} steps it outputs 1, and so on, changing from 0 to 1 or 1 to 0 every 2^{k-1} steps.

At time step 1 each base cell in the BPA constructs a state transition vector of length m based on the input value at the cell. Each non-base cell C behaves as follows: when C's counter is in state 0 (1) it constructs the composition of its upper-left (lower-left) and upper-right (lower-right) sons' transition vectors. Thus a cell in layer 1 alternately composes the transition vectors of its upper two and lower two sons. A cell in layer 2 composes vectors from its upper sons at the first two time steps (i.e., the first two rows of its base) and then at the next two steps computes the row scan transition vectors for its third and fourth rows. By induction, it follows that a cell in layer k computes left-to-right row scan transition vectors for the 2^k rows in its base in a top to bottom sequence at times $k,k+1,\ldots,k+2^k-1$; and this process repeats (modulo 2^k).

At the same time, cell C in layer k composes the 2^k row scan transition vectors to get a complete raster scan transition vector for C's base. This is accomplished by having C store a second cumulative transition vector. When C's counter switches from state 1 to state 0 we must be computing the vector for C's top row. Thus at this step we initialize the raster scan transition vector with the first row's vector. At all other times, we just compose the current row p's vector with the saved raster scan vector to obtain a new raster scan vector describing M's movement over the top p rows of input. In particular, at time $n+2^n-1$ the apex cell's raster scan transition vector describes the state that M ends in after scanning the base in raster order when starting in any of its m states. Thus if M's initial state gives rise to an accepting state in this vector, the BPA can accept, otherwise it rejects its input.

If the time required to look up a value in a table of length k is t(k), then $t(|Q_T|)$ is the time necessary for the base cells to initially set up their transition vectors. An additional $(n+2^n-1)t(m)$ time is then required before the apex cell can decide membership in $L(M)$. Thus $t(|Q_T|) + (n+2^n-1)t(m)$ or O(diameter) time is sufficient. This is faster than M itself, which requires O(area) time to scan its input. However, a CA can also simulate a raster scan FSA in O(diameter) time.

Similarly, we can simulate other fixed scanning sequences of an FSA, e.g., snake-like or column-major indexed scans, without significantly altering the construction given above. The critical knowledge that we have used here, which is not available in more general FSA's, is that the motion of

M is fixed in advance and so it is not necessary to save M's position on the tape.

We now return to the general problem: how does a BPA compare with an FSA which can move to any of its four adjacent neighbors during a transition? The problem is that M can enter or leave a given block of base cells at any place along its boundary, and so the description of M's behavior on a block grows with the size of the block. Consequently, we cannot specify M's behavior relative to an unbounded size base segment by a state transition vector of bounded length.

The search for an alternative method is worthwhile, however, since the following result shows that a BPA has sufficient time to distinguish between all possible base segments. Let m be the number of states in M and let $s \leq m$ be the number of input states. Then clearly the number of possible 2^k by 2^k input configurations is $D=s^{2^{2k}}$. From [1], we know that a cell in layer k of a BPA (with a 2^k by 2^k base segment) can have a state sequence period up to $m!(2^k)^{\log m}$. It follows that the total number of distinct sequences of states that a cell in layer k can have is bounded from above by $P=m^{m!(2^k)^{\log m}}$. However, when $k \gg m$ we have $D < P$, implying that a BPA's periodicity is not a limitation on distinguishing between all possible 2^k by 2^k input blocks.

In any event, the sets of languages accepted by BPS's and FSA's are not the same. The vertically symmetric palindromes were shown in Section 3.3 to be recognizable by a

BPA. Blum and Hewitt have proved [5] that this language cannot be accepted by an FSA.

4.3 UDPA's and PA's

UDPA's are a simplification of PA's in which the neighborhood definition eliminates all sidewise connections to brother cells. Though slower for some tasks, we show that UDPA's are equivalent to PA's. Clearly, any language that can be recognized by a UDPA can be recognized by a PA that ignores its brother links. We now show that any language recognized by a PA can also be recognized by a UDPA. We prove this by demonstrating how a UDPA can simulate a CA. For simplicity, we give the one-dimensional proof; the generalization to the two-dimensions is immediate.

Given a CA with input size N, let n be the smallest integer such that $2^n \geq N$. The input string will be left-justified in the base of the UDPA with the rightmost $2^n - N$ cells initialized to the boundary state #.

To simulate a single step of the CA, each UDPA base cell must have access to the states of its corresponding CA cell's two brother cells. Every two cells that are adjacent in the CA have a least common ancestor (LCA) in the UDPA. Furthermore, that UDPA cell which is the LCA for an adjacent pair in the CA cannot be the LCA for any other pair. Therefore, every two adjacent cells in the base of a UDPA can simultaneously switch state information as follows:

At time step 1, cells in layer 1 copy the pair of states in their base segments. At time step 2, left (right) sons in layer 0 copy the right (left) state stored in their father's cell. At the same time cells in layer 2 store the ordered quadruple of states in their base segments by copying the state pairs stored in layer 1 cells. Thus at the end of step 1 N/2 LCA's have been found, and at the end of step 2 each base cell knows the state of one of its brothers. At step 3, left (right) sons in layer 1 copy the third (second) state stored in their father's quadruple of states and mark it L(R). Also, cells in layer 3 compute their own ordered quadruple of states by copying the first and fourth members from their two son's quadruples. Thus at the end of step 2 N/4 more LCA's are found in layer 2. After step 3 the states of these N/4 adjacent cell pairs have been swapped by the sons of their LCA's. At subsequent steps, any right (left) son of a father with state marked L(R) copies its father's state. In this way the state information propagates back down the UDPA to the proper brother cell.

Similarly, state quadruples are computed higher and higher in the UDPA until every state pair's LCA is found. At time n the root is left with a pair of states corresponding to the states of the leftmost and rightmost cells in the base. Consequently, their brothers' states are the boundary state #, and this information can be returned to the proper base cell in the same manner. Thus simulating one step of a CA takes 2n or O(log diameter) time for the UDPA.

5. Conclusion

We have extended our study of pyramid cellular acceptors in this report, treating primarily the acceptance of two-dimensional properties by bottom-up pyramid acceptors. In particular, we have described how BPA's can do local property detection, but not counting. Since PA's can count local properties, the advantage of sidewise connections (at least in the base) has been established. In addition, we showed how a BPA can accept two-dimensional palindromes, a language which is not recognizable by any two-dimensional FSA. While PA's are shown to accept precisely the class of languages accepted by CA's, it is left open whether BPA's can accept *the two-dimensional finite state languages.*

Table I summarizes some of the BPA recognition results and open questions to date for one and two-dimensional languages.

References

1. C. R. Dyer and A. Rosenfeld, Cellular pyramids for image analysis, Technical Report 544, Computer Science Center, University of Maryland, College Park, Md., May, 1977.

2. M. Minsky and S. Papert, Perceptrons, MIT Press, Cambridge, Mass., 1969.

3. W. T. Beyer, Recognition of topological invariants by iterative arrays, MAC TR-66, Massachusetts Institute of Technology, October 1969.

4. S. Levialdi, On shrinking binary picture patterns, CACM 15, 1972, 7-10.

5. M. Blum and C. Hewitt, Automata on a 2-dimensional tape, Conference Record IEEE 8th Annual Symposium on Switching and Automata Theory, 1967, 155-160.

6. J. E. Hopcroft and J. D. Ullman, Formal languages and their relation to automata, Addison-Wesley, Reading, Mass., 1969.

TABLE I

Recognition Times for BPA's and BTA's

Language	BTA	BPA
1. Parity	O(log d)	O(log d)
2. Equality	O(log d)	O(log d)
3. Majority	O(log d)	O(log d)
4. Palindromes	O(d)	O(area)
5. Padded Palindromes	?	?
6. Dyck Language	O($\log^2$d)	-
7. Rectangles, Squares, Triangles, etc.	-	C(log d)
8. Local Property Detection	O(log d)	O(d)
9. Local Property Counting	O(log d)	No?
10. Connectedness	O(log d)	?
11. Convexity	O(log d)	?
12. Row-Column Convexity	O(log d)	O(d)
13. Finite State	O(log d)	?

Here d is the string length or array diameter. For CA's, the corresponding recognition times are all O(d).

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

18 19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 78-0126	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 CELLULAR PYRAMIDS FOR IMAGE ANALYSIS, 2.		5. TYPE OF REPORT & PERIOD COVERED 9 Interim rept.,
		6. PERFORMING ORG. REPORT NUMBER 14 TR-596
7. AUTHOR(s) 10 Charles R. Dyer		8. CONTRACT OR GRANT NUMBER(s) 15 AFOSR 77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Maryland Computer Science Center College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 16 17 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, DC 20332		12. REPORT DATE 11 ~~November~~ 1977
		13. NUMBER OF PAGES 12 38 p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

cellular automata
image analysis
pattern recognition
parallel processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In an earlier report, a new class of multilayer bounded cellular automata was defined and shown to improve the lower bound recognition time for many basic array recognition tasks. ~~We continue our~~ The investigation of pyramid cellular acceptors, by presenting new results on their capabilities as two-dimensional pattern-recognizing machines. is continued

403018